Docket No. 50277-2237 *Patent*

UNITED STATES PATENT APPLICATION

FOR

IN-PLACE EVOLUTION OF XML SCHEMAS

INVENTORS:

SAM IDICULA
SIVASANKARAN CHANDRASEKAR
NIPUN AGARWAL
RAVI MURTHY

PREPARED BY:

HICKMAN PALERMO TRUONG & BECKER LLP
1600 WILLOW STREET
SAN JOSE, CALIFORNIA 95125
(408) 414-1080

ASSIGNEE:

ORACLE INTERNATIONAL CORPORATION
500 ORACLE PARKWAY
REDWOOD SHORES, CA 94065

"Express Mail" mailing label number EV322192478US

Date of Deposit August 25, 2003

IN-PLACE EVOLUTION OF XML SCHEMAS

RELATED APPLICATIONS

[0001] The present application is related to the following U.S. Patent Applications, the entire contents of which are incorporated herein by reference for all purposes:

[0002] U.S. Patent Application Serial No. 10/260,138, filed on September 27, 2002, entitled OPERATORS FOR ACCESSING HIERARCHICAL DATA IN A RELATIONAL SYSTEM, by Nipun Agarwal, Ravi Murthy, Eric Sedlar, Sivasankaran Chandrasekar and Fei Ge;

[0003] U.S. Patent Application Serial No. 10/260,384, filed on September 27, 2002, entitled PROVIDING A CONSISTENT HIERARCHICAL ABSTRACTION OF RELATIONAL DATA, by Nipun Agarwal, Eric Sedlar, Ravi Murthy and Namit Jain;

[0004] U.S. Patent Application Serial No. 10/259,278, filed on September 27, 2002, entitled MECHANISM FOR MAPPING XML SCHEMAS TO OBJECT-RELATIONAL DATABASE SYSTEMS, by Ravi Murthy, Muralidhar Krishnaprasad, Sivasankaran Chandrasekar, Eric Sedlar, Vishu Krishnamurthy and Nipun Agarwal;

[0005] U.S. Patent Application Serial No. 10/260,161, filed on September 27, 2002, entitled INDEXING TO EFFICIENTLY MANAGE VERSIONED DATA IN A DATABASE SYSTEM , by Nipun Agarwal, Eric Sedlar and Ravi Murthy;

[0006] U.S. Patent Application Serial No. 10/256,524, filed on September 27, 2002, entitled MECHANISMS FOR STORING CONTENT AND PROPERTIES OF HIERARCHICALLY ORGANIZED RESOURCES, by Ravi Murthy, Eric Sedlar, Nipun Agarwal, and Neema Jalali;

[0007] U.S. Patent Application Serial No. 10/259,176, filed on September 27, 2002, entitled MECHANISM FOR UNIFORM ACCESS CONTROL IN A DATABASE SYSTEM, by Ravi Murthy, Eric Sedlar, Nipun Agarwal, Sam Idicula, and Nicolas Montoya;

[0008] U.S. Patent Application Serial No. 10/256,777, filed on September 27, 2002, entitled LOADABLE UNITS FOR LAZY MANIFESTATION OF XML DOCUMENTS by Syam Pannala, Eric Sedlar, Bhushan Khaladkar, Ravi Murthy, Sivasankaran Chandrasekar, and Nipun Agarwal;

[0009] U.S. Patent Application Serial No. 10/260,381, filed on September 27, 2002, entitled MECHANISM TO EFFICIENTLY INDEX STRUCTURED DATA THAT PROVIDES HIERARCHICAL ACCESS IN A RELATIONAL DATABASE SYSTEM, by Neema Jalali, Eric Sedlar, Nipun Agarwal, and Ravi Murthy;

[0010] U.S. Patent Application Serial No. _____, filed on the same day herewith, entitled DIRECT LOADING OF SEMISTRUCTURED DATA, by Namit Jain, Nipun Agarwal, and Ravi Murthy (Attorney Docket No. 50277-2235);

[0011] U.S. Patent Application Serial No. _____, filed on the same day herewith, entitled DIRECT LOADING OF OPAQUE TYPES, by Namit Jain, Ellen Batbouta, Ravi Murthy, Nipun Agarwal, Paul Reilly, and James Stenoish (Attorney Docket No. 50277-2236); and

[0012] U.S. Patent Application Serial No. _____, filed on the same day herewith, entitled MECHANISM TO ENABLE EVOLVING XML SCHEMA, by Sam Idicula, Nipun Agarwal, Ravi Murthy, Eric Sedlar, and Sivasankaran Chandrasekar (Attorney Docket No. 50277-2238).

FIELD OF THE INVENTION

[0013] The present invention relates to data management systems, and in particular, to techniques for updating an XML schema, and for updating XML-schema-based instance documents and database structures to conform to an updated XML schema.

BACKGROUND OF THE INVENTION

[0014] Using Extensible Markup Language (XML), information may be represented in conformity with a specified hierarchical structure. An XML schema defines such a structure. An XML schema comprises a root XML element. One or more other XML elements may be nested within the root XML element as content of the root XML element. Such nested XML elements are called child XML elements of the root XML element. Conversely, the root XML element is called the parent XML element of the child XML elements. Each child XML element may, in turn, be a parent XML element of one or more additional child XML elements nested within the parent XML element. The parent-child relationships of XML elements within an XML schema define a hierarchical structure according to which information may be stored in a hierarchically structured manner. For each XML element in an XML schema, the XML schema defines the type of that XML element's content value.

[0015] Information stored in conformity with an XML schema does not need to indicate the XML tags of the XML elements in the XML schema. Specifically, as long as it is known to which XML schema such information conforms, content values that correspond to XML elements in an XML schema may be stored without the XML tags that would enclose the content values. For example, content values may be stored according to a format in which the content values are separated by delimiters such as comma characters. Reference may be made to the corresponding XML schema in order to align the content values with their corresponding XML elements in the XML schema.

[0016] For another example, database structures, such as database tables and database views, may be generated based on an XML schema. The names and data types associated with columns in such database tables may correspond to names and data types indicated by attributes of XML elements in the XML schema. A content value that corresponds to a particular XML element in the XML schema may be stored in a database table's column that corresponds to the particular XML element.

[0017] Multiple different sets of content values, each based on the same XML schema, may be stored distinctly from each other. Each set is a separate "instance document." For example, an XML schema may define an XML element such as "<element name='quantity' type='integer'>". One instance document may contain a content value of "1" that corresponds to the XML element. Another instance document may contain a content value of "2" that corresponds to the same XML element. When content values are stored in database tables, content values from different instance documents may be stored in different rows of the database tables. Content values that correspond to the same XML element may be stored in the same column of a database table.

[0018] Often, even after many instance documents have been generated in conformity with a particular XML schema, it may be desirable to evolve the XML schema. For example, it may be desirable to add a new XML element to the XML schema, sometimes by inserting the new XML element between existing XML elements in the XML schema. For another example, if a particular XML element represents an enumerated data type, it may be desirable to insert a new XML element into a set of child XML elements of the particular XML element, where each child XML element represents a different enumerated value.

[0019] An XML schema may be modified manually using, for example, a text editing tool. Unfortunately, as a consequence of the modification of the XML schema, existing

instance documents that formerly conformed to the XML schema might cease to conform to the XML schema. Instance documents interpreted according to the modified XML schema might be interpreted incorrectly, causing content values in the instance documents to be aligned with the wrong XML elements in the modified XML schema.

[0020] Additionally, database structures, such as database tables and database views, whose structures were based on the XML schema prior to the modification of the XML schema, might also cease to conform to the XML schema. As a result, it might be impossible to store correctly, in such database structures, content values in instance documents generated in conformity with the modified XML schema. For example, a database table might lack a column that corresponds to a new XML element that was inserted into the XML schema. Furthermore, existing instance documents may be interpreted incorrectly if the mapping of a specific column to a specific XML element has changed.

[0021] There is no established approach for ensuring that XML-schema-based instance documents and database structures will continue to conform to the XML schema upon which they are based after the XML schema has been modified. A technique is needed for evolving an XML schema while ensuring that XML-schema-based instance documents and database structures will continue to conform to the XML schema even after the XML schema has been evolved.

[0022] The approaches described in this section are approaches that could be pursued, but not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated, it should not be assumed that any of the approaches described in this section qualify as prior art merely by virtue of their inclusion in this section.

BRIEF DESCRIPTION OF THE DRAWINGS

[0023] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0024] Figure 1 is a block diagram that illustrates the interaction between structures involved in in-place XML schema evolution, according to an embodiment of the present invention;

[0025] Figure 2 is a flow diagram that illustrates a technique, according to an embodiment of the present invention, for evolving an XML schema and XML-schema-based database structures in place; and

[0026] Figure 3 is a block diagram that illustrates a computer system upon which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION OF THE INVENTION

[0027] A method and system are provided for in-place evolution of (1) an XML schema, (2) database object types and tables that are based on the XML schema, and (3) database object instances that are based on the database object types. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

FUNCTIONAL OVERVIEW

[0028] It is desirable that instance documents and database structures that conform to an XML schema evolve with the XML schema so that the instance documents and database structures continue conform to the XML schema after the XML schema has been evolved. According to one embodiment of the present invention, such continued conformity is achieved with a schema evolver that executes in a computer system.

[0029] The schema evolver receives both an existing XML schema and an XML document as input. The XML document indicates one or more changes to be made to the existing XML schema. Based on the existing XML schema and the XML document, the schema evolver first generates a new XML schema that incorporates the changes indicated in the XML document.

[0030] Additionally, according to one embodiment, the schema evolver generates one or more Structured Query Language (SQL) statements based on the new XML schema. The SQL statements, when executed by a database server, cause the database server to evolve

database structures that were based on the formerly existing XML schema so that the database structures conform to the new XML schema. Such database structures may define, for example, database object types and database object instances.

DATABASE OBJECT TYPES, DATABASE OBJECT TABLES, AND DATABASE OBJECT INSTANCES

[0031] A database object type specifies the structure and attributes of all database object instances that are based on the database object type. For example, an "address" object type might specify a sequence of three primitive "VARCHAR" object types: one to store a person's name, one to store a street identifier, and one to store a city identifier. A database object table is a database table that is based on a database object type. For example, an "address" object table based on the "address" object type would comprise at least three separate columns; one for a person's name, one for a street identifier, and one for a city identifier. When a database object type is altered, database object tables based on that database object type are altered correspondingly.

[0032] Based on the "address" object type, multiple different "address" object instances may be created. Each of the "address" object instances conforms to the structure and attributes specified by the "address" object type. Each different "address" object instance may specify a different value for a particular attribute specified by the "address" object. For example, one "address" object instance may specify "Joe Smith" as the value of the name attribute, while another "address" object instance may specify "John Taylor" as the value of the name attribute.

[0033] When an XML schema is evolved, XML elements may be added to or deleted from the XML schema. The addition of a new XML element may cause the schema evolver

to generate an SQL statement that, when executed by a database server, causes the database server to add a corresponding attribute to a database object type. The SQL statement, when executed by the database server, may also cause the database server to add a corresponding column to a database table that stores database object instances that are based on the database object type.

[0034] Similarly, the deletion of an XML element may cause the schema evolver to generate an SQL statement that, when executed by the database server, causes the database server to remove a corresponding attribute from a database object type. The SQL statement, when executed by the database server, may also cause the database server to drop a corresponding column from a database table that stores database object instances that are based on the database object type.

[0035] Thus, the SQL statements generated by the schema evolver may cause both database object types and database object instances to be evolved to conform to a new XML schema. Such SQL statements may cause database object types and database object instances to be created, deleted, and/or altered.

IN-PLACE EVOLUTION

[0036] Techniques described herein allow for "in-place" evolution of XML schemas, as well as database object types and database object instances that are based on such XML schemas. Through "in-place" evolution, additional copies of information represented in such structures do not need to be made. Instead, the information represented by such structures may be updated without making additional copies of the information. Changes to an existing XML schema may be made automatically by modifying the XML schema according to changes indicated by an XML document. A database server may execute automatically

generated SQL statements that cause the database server to alter existing database object types and database object instances to conform to the modified XML schema.

[0037] "In-place" evolution differs from "data copy" evolution. "Data copy" evolution involves making a copy of the information represented by the structures to be evolved, then the deleting the structures, then creating new structures, and then inserting the information from the copy back into corresponding components of the new structures. "In-place" evolution typically exhibits performance superior to "data copy" evolution and does not require the disk space that data copy evolution requires.

INTERACTION BETWEEN STRUCTURES INVOLVED IN IN-PLACE XML SCHEMA EVOLUTION

[0038] Figure 1 is a block diagram that illustrates the interaction between structures involved in in-place XML schema evolution, according to an embodiment of the present invention. A schema evolver 102 is a component of a database server 104. Database server 104 executes in a computer system. The schema evolver 102 receives, as input, an existing XML schema 106 and an XML document 108. Existing XML schema 106 is an XML schema that is currently registered in database 110. XML document 108 indicates, in XML, changes to be made to existing XML schema 106.

[0039] Schema evolver 102 automatically alters existing XML schema 106 so that the changes indicated by XML document 108 are incorporated into existing XML schema 106. As a result, existing XML schema 106 evolves into evolved XML schema 112. Evolved XML schema 112 incorporates all of the changes indicated by XML document 108.

[0040] According to one embodiment, schema evolver 102 receives, as input, evolved XML schema 112, existing database object types 114, and existing database object instances

116. Existing database object types 114 conform to the structure indicated by existing XML schema 106. Existing database object instances 116 conform to the type definitions indicated by database object types 114.

[0041] Schema evolver 102 determines which aspects of existing database object types 114 do not conform to evolved XML schema 112. Based on these non-conforming aspects, schema evolver 102 automatically generates SQL statements 118 that, when executed by database server 104, cause database server 104 to alter existing database object types 114 to conform to evolved XML schema 112. Additionally, when executed by database server 104, SQL statements 118 also cause the database server to make existing database object instances 116 conform to the changes to existing type definitions 114 that will result from database server 104 executing the SQL statements.

[0042] Database server 104 executes SQL statements 118. As a result, database server 104 evolves existing database object types 114 into evolved database object types 120. As another result, database server 104 also evolves existing database object instances 116 into evolved database object instances 122. Evolved database object types 120 conform to the structure indicated by evolved XML schema 112. Evolved database object instances 122 conform to the structures indicated by evolved database object types 120.

ROLLBACK STATEMENTS

[0043] SQL statements 118 may be called "evolve" statements because, when executed, the SQL statements cause database server 104 to evolve existing database object types 114 and existing database object instances 116. In order to make the evolution an atomic transaction, schema evolver 102 also generates SQL statements 124 that, if executed by database server 104, will cause the database server to undo, or "roll back" all of the effects of

SQL statements 118 that have been executed when the error occurs. Therefore, SQL statements 124 may be called "rollback statements."

[0044] If, during execution of SQL statements 118, an error occurs, then database server 104 executes those of SQL statements 124 that will reverse the effects of those of SQL statements 118 that the database server has executed. Therefore, unless the entire evolution completes successfully, no part of the evolution is made permanent.

[0045] Changes to XML schema 106 are also reversed when an error occurs during XML schema evolution. For example, schema evolver 102 may track changes that schema evolver 102 makes to XML schema 106 and then undo those changes if an error occurs before the entire evolution transaction is complete.

EXAMPLE TECHNIQUE FOR EVOLVING AN XML SCHEMA AND XML-SCHEMA-BASED DATABASE STRUCTURES IN PLACE

[0046] Figure 2 is a flow diagram that illustrates a technique 200, according to an embodiment of the present invention, for evolving an XML schema and XML-schema-based database structures in place. In block 202, a schema evolver receives an XML document. The XML document indicates one or more changes to be made to an existing XML schema.

[0047] In block 204, the schema evolver generates, based on the existing XML schema and the XML document, an evolved XML schema. The evolved XML schema incorporates all of the changes indicated by the XML document. The schema evolver may track the changes that were made to the existing XML schema in order to produce the evolved XML schema.

[0048] In block 206, the schema evolver generates, based on the evolved XML schema, "evolve" SQL statements. The "evolve" SQL statements, when executed by a database

server, cause the database server to do one or more of the following: create a new database structure, delete an existing database structure, and/or alter an existing database structure. Database structures that may be created, deleted, and/or altered may include structures that define database object types and structures that define database object instances.

[0049] The "evolve" SQL statements, when executed by a database server, cause the database server to make database object types that are based on the pre-evolved XML schema conform to the structure indicated by the evolved XML schema. Thus, the database object types are evolved. Also, when a database server executes the "evolve" SQL statements, the "evolve" SQL statements cause the database server to make database object instances that are based on the pre-evolved database object types conform to the evolved database object types. Thus, the database object instances are evolved.

[0050] For example, the schema evolver may generate, based on an evolved XML schema that incorporates a new XML element, an SQL statement that causes a database server to add a new column to a database table. The new column may have the same name as the name indicated by the attributes of the new XML element, and may be populated with default values based on the type indicated by the attributes of the new XML element. SQL statements that the schema evolver may generate may include actions such as "CREATE," "ALTER," "ADD," "MODIFY," and "DROP." Such SQL statements may indicate whether the target of the SQL statement is a "TYPE," a "TABLE," etc.

[0051] In block 208, the schema evolver generates one or more "rollback" SQL statements. The schema evolver may generate the "rollback" SQL statements as the schema evolver generates the "evolve" SQL statements. The "rollback" SQL statements, when executed by a database server, cause the database server to reverse the effects of the "evolve" SQL statements on the targets of the "evolve" SQL statements.

[0052] In block 210, the database server executes the "evolve" SQL statements. As a result, database object types and database object instances may be created, deleted, or altered. If no error occurs, then the resulting database object types and database object instances conform to the evolved XML schema.

[0053] While the database server executes the "evolve" SQL statements, it is determined, in block 212, whether an error has occurred with regard to the execution of the "evolve" SQL statements. If the "evolve" SQL statements execute entirely without an error, then control passes to block 214. Otherwise, control passes to block 216 as soon as the error occurs.

[0054] If no error occurs, then, in block 214, the evolution is finished. However, if an error occurs during the execution of the "evolve" SQL statements, then, in block 216, the database server executes as many of the "rollback" SQL statements as are necessary to reverse the effects of the already executed "evolve" SQL statements on the targets of the "evolve" SQL statements. The schema evolver may also undo the changes made to the XML schema based on the changes the schema evolver tracked while evolving the XML schema. As a result, the XML schema and database structures based on the XML schema are rolled back to their pre-evolutionary state. Thus, the evolution is guaranteed to be an atomic operation.

THE XDIFF SCHEMA

[0055] As described above, a schema evolver may evolve an existing XML schema into an evolved XML schema based on an XML document that indicates changes to be made to the existing XML schema. The XML document may express the changes in terms of XML elements defined by an "xdiff" XML schema of which the XML document is an instance document.

[0056] The xdiff schema defines, in XML, primitive elements through which a user may express evolution commands. Such commands may express operations relative to "nodes." As used herein, a "node" is some aspect of an XML structure such as an XML element. For example, one element of the xdiff schema may define the structure of a command to append a specified node to the end of a specified list of nodes. Another element of the xdiff schema may define the structure of a command to insert a specified node immediately before another specified node. Yet another element of the xdiff schema may define the structure of a command to delete a specified node and the child nodes of the specified node.

[0057] An exemplary xdiff schema is shown below:

```
<schema targetNamespace="http://xmlns.company.com/xdb/xdiff.xsd"
  xmlns="http://www.w3.org/2001/XMLschema"
  xmlns="http://www.xmlns.company.com/xdb/xdiff.xsd"
  version="1.0" elementFormDefault="qualified">
  <simpleType name="xdiff-nodetype">
    <restriction base="string">
      <enumeration value="element"/>
      <enumeration value="attribute"/>
      <enumeration value="text"/>
      <enumeration value="cdata"/>
      <enumeration value="entity-reference"/>
      <enumeration value="entity"/>
      <enumeration value="processing-instruction"/>
      <enumeration value="notation"/>
    </restriction>
```

```
</simpleType>
<element name="xdiff">
  <complexType>
    <choice maxOccurs="unbounded">
      <element name="append-node">
        <complexType>
          <sequence>
            <element name="content" type="anyType"/>
          </sequence>
          <attribute name="parent-xpath" type="string"/>
          <attribute name="node-type" type="xd:xdiff-nodetype/>
        </complexType>
      </element>
      <element name="insert-node-before">
        <complexType>
          <sequence>
            <element name="content" type="anyType"/>
          </sequence>
          <attribute name="xpath" type="string"/>
          <attribute name="node-type" type="xd:xdiff-nodetype/>
        </complexType>
      </element>
      <element name="delete-node">
        <complexType>
```

```
        <attribute name="xpath" type="string"/>
      </complexType>
    </element>
  </choice>
 </complexType>
</element>
</schema>
```

[0058] In the above xdiff schema, three elements are defined: "append-node," "insert-node-before," and "delete-node." The "append-node" element expresses syntax for indicating that the schema evolver should add, as the last child node of the node specified by the "parent-xpath" attribute, the node specified by the "content" element. The node specified by the "content" element is indicated to be of a node type specified by the "node-type" element.

[0059] For example, an XML document provided to the schema evolver might contain an element such as:

```
<xd:append-node parent-xpath="/schema/simpleType/restriction" node-
  type="element">
  <xd:content>
    <enumeration value="FL"/>
  </xd:content>
</xd:append-node>
```

Based on this, the schema evolver would add "<enumeration value='FL'/>" to the below XML schema, as shown after the addition:

```
<schema targetNamespace="http://www.company.com/po.xsd">
```

```
<simpleType name="USState">
  <restriction base="string">
    <enumeration value="NY"/>
    <enumeration value="CA"/>
    <enumeration value="FL"/>
  </restriction>
</simpleType>
</schema>
```

[0060] The "insert-node-before" element expresses syntax for indicating that the schema evolver should insert the node specified by the "content" element immediately before the node specified by the "xpath" attribute. Again, the node specified by the "content" element is indicated to be of a node type specified by the "node-type" element.

[0061] For example, an XML document provided to the schema evolver might contain an element such as:

```
<xd:insert-node-before xpath="/schema/simpleType" node-type="comment">
  <xd:content>
    <!-- A type representing US States -->
  </xd:content>
</xd:insert-node-before>
```

Based on this, the schema evolver would insert "<!-- A type representing US States -->" in the below XML schema, as shown after the insertion:

```
<schema targetNamespace="http://www.company.com/po.xsd">
  <!-- A type representing US States -->
  <simpleType name="USState">
```

```
        <restriction base="string">
          <enumeration value="NY"/>
          <enumeration value="CA"/>
        </restriction>
      </simpleType>
    </schema>
```

[0062] The "delete-node" element expresses syntax for indicating that the schema evolver should delete the node specified by the "xpath" attribute, along with all of the child nodes of that node. In combination with the "append-node" or "insert-node-before" elements, the "delete-node" element may be used to express a modification to an existing node in an XML schema.

SPECIFYING NODES IN XPATH

[0063] Because different nodes at different levels of a hierarchy may be expressed by identical text at different locations in an XML schema, merely searching for one specified text string in the XML schema and replacing the text string with another specified text string may not result in a correct transformation. Therefore, XML Path Language (XPath) expressions are used, in an XML document, to specify nodes in an XML schema precisely. An XPath expression identifies a node relative to its location in a hierarchy.

[0064] For example, an XML hierarchy might indicate the following nodes:

```
    <foo1 xmlns:nmsp1="company">
      <nmsp1:foo1>2</nmsp1:foo1>
      <foo1><foo2>23</foo2></foo1>
    </foo1>
```

The XPath expression "/foo1/nmsp1:foo1" precisely identifies the element that contains the content value "2."

HARDWARE OVERVIEW

[0065] Figure 3 is a block diagram that illustrates a computer system 300 upon which an embodiment of the invention may be implemented. Computer system 300 includes a bus 302 or other communication mechanism for communicating information, and a processor 304 coupled with bus 302 for processing information. Computer system 300 also includes a main memory 306, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 302 for storing information and instructions to be executed by processor 304. Main memory 306 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 304. Computer system 300 further includes a read only memory (ROM) 308 or other static storage device coupled to bus 302 for storing static information and instructions for processor 304. A storage device 310, such as a magnetic disk or optical disk, is provided and coupled to bus 302 for storing information and instructions.

[0066] Computer system 300 may be coupled via bus 302 to a display 312, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 314, including alphanumeric and other keys, is coupled to bus 302 for communicating information and command selections to processor 304. Another type of user input device is cursor control 316, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 304 and for controlling cursor movement on display 312. This input device typically has two degrees of freedom in two

axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0067] The invention is related to the use of computer system 300 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system 300 in response to processor 304 executing one or more sequences of one or more instructions contained in main memory 306. Such instructions may be read into main memory 306 from another computer-readable medium, such as storage device 310. Execution of the sequences of instructions contained in main memory 306 causes processor 304 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0068] The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 304 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 310. Volatile media includes dynamic memory, such as main memory 306. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 302. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0069] Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a

RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0070] Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 304 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 300 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 302. Bus 302 carries the data to main memory 306, from which processor 304 retrieves and executes the instructions. The instructions received by main memory 306 may optionally be stored on storage device 310 either before or after execution by processor 304.

[0071] Computer system 300 also includes a communication interface 318 coupled to bus 302. Communication interface 318 provides a two-way data communication coupling to a network link 320 that is connected to a local network 322. For example, communication interface 318 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 318 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 318 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0072] Network link 320 typically provides data communication through one or more networks to other data devices. For example, network link 320 may provide a connection through local network 322 to a host computer 324 or to data equipment operated by an Internet Service Provider (ISP) 326. ISP 326 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 328. Local network 322 and Internet 328 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 320 and through communication interface 318, which carry the digital data to and from computer system 300, are exemplary forms of carrier waves transporting the information.

[0073] Computer system 300 can send messages and receive data, including program code, through the network(s), network link 320 and communication interface 318. In the Internet example, a server 330 might transmit a requested code for an application program through Internet 328, ISP 326, local network 322 and communication interface 318.

[0074] The received code may be executed by processor 304 as it is received, and/or stored in storage device 310, or other non-volatile storage for later execution. In this manner, computer system 300 may obtain application code in the form of a carrier wave.

[0075] In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. Thus, the sole and exclusive indicator of what is the invention, and is intended by the applicants to be the invention, is the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction. Any definitions expressly set forth herein for terms contained in such claims shall govern the meaning of such terms as used in the claims. Hence, no limitation, element,

property, feature, advantage or attribute that is not expressly recited in a claim should limit the scope of such claim in any way. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.